珠峰架构课 Vue3组件库搭建

一.搭建monorepo环境

> 使用pnpm安装包速度快，磁盘空间利用率高效，使用pnpm可以快速建立monorepo，so ~ 这里我们使用pnpm workspace来实现monorepo

```
npm install pnpm -g # 全局安装pnpm
pnpm init # 初始化package.json配置文件
pnpm install vue@next typescript -D 全局下添加依赖
npx tsc --init # 初始化ts配置文件
```

> 使用pnpm必须要建立.npmrc文件，shamefully-hoist = true，否则安装的模块无法放置到node_modules目录下

```
{
  "compilerOptions": {
    "module": "ESNext", // 打包模块类型ESNext
    "declaration": false, // 默认不要声明文件
    "noImplicitAny": false, // 支持类型不标注可以默认any
    "removeComments": true, // 删除注释
    "moduleResolution": "node", // 按照node模块来解析
    "esModuleInterop": true, // 支持es6,commonjs模块
    "jsx": "preserve", // jsx 不转
    "noLib": false, // 不处理类库
    "target": "es6", // 遵循es6版本
    "sourceMap": true,
    "lib": [ // 编译时用的库
      "ESNext",
      "DOM"
    ],
    "allowSyntheticDefaultImports": true, // 允许没有导出的模块中导入
    "experimentalDecorators": true, // 装饰器语法
    "forceConsistentCasingInFileNames": true, // 强制区分大小写
    "resolveJsonModule": true, // 解析json模块
    "strict": true, // 是否启动严格模式
    "skipLibCheck": true // 跳过类库检测
  },
  "exclude": [ // 排除掉哪些类库
    "node_modules",
    "**/__tests__",
    "dist/**"
  ]
}
```

在项目根目录下建立pnpm-workspace.yaml 配置文件

```
packages:
  - 'packages/**' # 存放编写组件的
  - docs # 存放文档的
  - play # 测试组件的
```

二.创建组件测试环境

```
mkdir play && cd play
pnpm init
pnpm install vite @vitejs/plugin-vue # 安装vite及插件
```

```
{
  "name": "@z-plus/play",
  "private":true,
  "scripts": {
    "dev": "vite"
  }
}
```

vite.config.ts

```
import {defineConfig} from 'vite'
import vue from '@vitejs/plugin-vue';

export default defineConfig({
    plugins:[vue()]
})
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
    <div id="app"></div>
    <script type="module" src="/main.ts"></script>
</body>
</html>
```

main.ts

```
import {createApp} from 'vue';
import App from './app.vue'
const app = createApp(App) // 编写play组件为测试入口
app.mount('#app')
```

> 提供typescript声明文件 **typings/vue-shim.d.ts**

```
declare module '*.vue'{
    import type {DefineComponent} from 'vue'
    const component:DefineComponent
    export default component
}
```

三.编写Icon组件

```
packages
    ├─components  # 存放所有的组件
    ├─utils       # 存放工具方法
    └─theme-chalk # 存放对应的样式
```

1.模块间的相互引用

```
cd components && pnpm init # @z-plus/components
cd utils && pnpm init # @z-plus/utils
cd theme-chalk && pnpm init # @z-plus/theme-chalk
```

> 在根模块下添加依赖

```
pnpm install @z-plus/components -w
pnpm install @z-plus/theme-chalk -w
pnpm install @z-plus/utils -w
```

2.实现Icon组件

> 新增icon组件 **components/icon/src/icon.ts**

```
import { ExtractPropTypes } from "vue"
export const iconProps = {
    size: {
        type: Number
    },
    color: {
        type: String
    }
} as const
export type IconProps = ExtractPropTypes<typeof iconProps>
```

> icon组件编写 **components/icon/src/icon.vue**

```
<template>
    <i class="z-icon" :style="style">
        <slot></slot>
    </i>
</template>
<script lang="ts">
import { computed, defineComponent } from "vue";
import type {CSSProperties} from 'vue'
import {iconProps} from './icon'
export default defineComponent({
    name:'ZIcon',
```

```
    props: iconProps,
    setup(props){
        const style = computed<CSSProperties>(()=>{
            if(!props.size && !props.color){
                return {}
            }
            return {
                ...(props.size ? {'font-size': props.size+'px'}:{}),
                ...(props.color ? {'color': props.color}:{})
            }
        });
        return {
            style
        }
    }
})
</script>
```

2.导出Icon组件

> icon组件入口 **components/icon/index.ts**

```
import Icon from "./src/icon.vue";
import { Plugin, App } from "vue";
type SFCWithInstall<T> = T & Plugin;
const withInstall = <T>(comp: T) => {
  (comp as SFCWithInstall<T>).install = function (app: App):void {
    app.component((comp as any).name, comp);
  };
  return comp as SFCWithInstall<T>;
};
export const ZIcon = withInstall(Icon);
export default ZIcon; // 导出组件
export * from "./src/icon"; // 导出组件的属性类型
```

每个组件都需要增添install方法，我们将`withInstall`方法拿到`utils`中

```
import { Plugin,App } from 'vue'
export type SFCWithInstall<T> = T & Plugin;
export const withInstall = <T>(comp: T) => {
    (comp as SFCWithInstall<T>).install = function (app: App) {
        app.component((comp as any).name, comp)
    }
    return comp as SFCWithInstall<T>
}
```

```
import Icon from './src/icon.vue';
import { withInstall } from '@z-plus/utils/with-install';
export const ZIcon = withInstall(Icon);
export default ZIcon;       // 导出组件
export * from './src/icon';// 导出组件的属性类型
```

> 这样我们就可以在`components`下使用utils模块了。

4.展示组件

```
<template>
    <ZIcon :size="20">hello</ZIcon>
</template>
<script setup lang="ts">
import { ZIcon } from '@z-plus/components/icon'
</script>
```

四.字体图标

iconfont 更改FontClass前缀`z-icon-`，更改FontFamily为`z-ui-icons`，下载字体图标到`fonts`目录下

```
theme-chalk
|  └─src
|      ├─fonts  # 存放字体
|      └─mixins
|           └─config.scss # BEM规范命名
```

> mixins/config.scss

```
$namespace: 'z';
$element-separator: '__';
$modifier-separator:'--';
$state-prefix:'is-';
```

> mixins/mixins.scss

```
@use 'config' as *;
@forward 'config';
```

> icon.scss

```
@use 'mixins/mixins' as *;
@font-face {
    font-family: "z-ui-icons"; /* Project id 2856905 */
    src: url('./fonts/iconfont.woff2') format('woff2'),
        url('./fonts/iconfont.woff') format('woff'),
        url('./fonts/iconfont.ttf') format('truetype');
}

[class^='#{$namespace}-icon'],[class*=' #{$namespace}-icon']{
    font-family: "z-ui-icons" !important;
    font-size: 16px;
    font-style: normal;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
```

> index.scss

```
@use './icon.scss';
```

五.打包组件库

> 我们整个打包流程可以通过`gulp`来进行流程控制

```
"scripts": {
    "build": "gulp -f build/gulpfile.ts"
}
```

```
pnpm install gulp @types/gulp sucrase -w -D
```

1.`gulp`控制打包流程

```
import { series, parallel } from 'gulp';
import { withTaskName, run } from './utils'
export default series(
    withTaskName('clean', () => run('rm -rf ./dist')),
)
```

`build/utils/paths.ts`

```
import path from 'path'
export const projectRoot = path.resolve(__dirname, "../../");
```

`build/utils/index.ts`

```
import { spawn } from 'child_process';
import { projectRoot } from './paths';
export const withTaskName = <T>(name: string, fn: T) => Object.assign(fn, {
displayName: name });
export const run = async (command: string) => {
    return new Promise((resolve) => {
        const [cmd, ...args] = command.split(' ');
        const app = spawn(cmd, args, {
            cwd: projectRoot,
            stdio: 'inherit',
            shell: true
        });
        app.on('close', resolve)
    })
}
```

2.打包`scss`样式

```
export default series(
    withTaskName('clean', () => run('rm -rf ./dist')),
    parallel(
        withTaskName("buildPackages", () =>
            run("pnpm run --filter ./packages --parallel build")
        )
    )
)
```

> 会依次调用packages目录下对应包的build命令

```
"scripts": {
    "build": "gulp"
}
```

```
pnpm install gulp-sass @types/gulp-sass @types/sass @types/gulp-autoprefixer
gulp-autoprefixer @types/gulp-clean-css gulp-clean-css sass -D -w
```

> `theme-chalk/gulpfile.ts`

```
import { series, src, dest } from 'gulp'
import gulpSass from 'gulp-sass' // 处理sass
import dartSass from 'sass'
import autoprefixer from 'gulp-autoprefixer' // 添加前缀
import cleanCSS from 'gulp-clean-css' // 压缩css
import path from 'path'

function compile() { // 处理scss文件
    const sass = gulpSass(dartSass)
    return src(path.resolve(__dirname, './src/*.scss'))
        .pipe(sass.sync())
        .pipe(autoprefixer({}))
        .pipe(cleanCSS())
        .pipe(dest('./dist/css'))
}
function copyfont() { // 拷贝字体样式
    return src(path.resolve(__dirname,
'./src/fonts/**')).pipe(cleanCSS()).pipe(dest('./dist/fonts'))
}
function copyStyle(){
    return
src(path.resolve(__dirname,'dist/**')).pipe(dest(path.resolve(__dirname,'../../d
ist/theme-chalk')));
}
const buildStyle = series(
    compile,
    copyfont,
    copyStyle
)
export default buildStyle
```

3.打包工具模块

> `utils/gulpfile.ts`

```
import { buildPackage } from '../../build/packages'
export default buildPackage(__dirname, 'utils') // 打包这个模块
```

```
pnpm install gulp-typescript -w -D
```

> `build/utils/config.ts`

```
```

```
import path from "path";
import { outDir } from "./paths";
export const buildConfig = {
  esm: {
    module: "ESNext",
    format: "esm",
    output: {
      name: "es",
      path: path.resolve(outDir, "es"),
    },
    bundle: {
      path: "z-plus/es",
    },
  },
  cjs: {
    module: "CommonJS",
    format: "cjs",
    output: {
      name: "lib",
      path: path.resolve(outDir, "lib"),
    },
    bundle: {
      path: "z-plus/lib",
    },
  },
};
export type BuildConfig = typeof buildConfig;
```

build/packages.ts

```
import { parallel, series, src, dest } from "gulp";
import path from "path";
import ts from "gulp-typescript";
import { buildConfig } from "./utils/config";
import { outDir, projectRoot } from "./utils/paths";
import { withTaskName } from "./utils";
export const buildPackage = (pkgPath: string, packageName: string) => {
    const tasks = Object.entries(buildConfig).map(([module, config]) => {
        const output = path.resolve(pkgPath, config.output.name);
        return series(
            withTaskName(`build:${packageName}`, () => {
                const tsConfig = path.resolve(projectRoot, "tsconfig.json");
                const inputs = ["**/*.ts", "!gulpfile.ts", "!node_modules"];
                return src(inputs)
                    .pipe(
                        ts.createProject(tsConfig, {
                            module: config.module,
                            declaration: true,
                            moduleResolution: "node",
                            strict: false,
                        })()
                    )
                    .pipe(dest(output));
            }),
            withTaskName(`copy:${packageName}`, () => {
                return src(`${output}/**`).pipe(
                    dest(
```

```
                        path.resolve(outDir, config.output.name, packageName)
                    )
                );
            })
        );
    });
    return parallel(...tasks);
};
```

###